ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Tarefa 3 - MAP3121

Modelagem de um Sistema de Resfriamento de Chips

Arthur de Azevedo e Almeida Maia — 11819921
Caio Nascimento Balreira — 11805342

São Paulo, SP
2022

1. Introdução

Nesta tarefa, estuda-se como modelar o comportamento da difusão térmica que ocorre em um processador ou chip de computador de tamanho L × L e altura h ao usarmos um resfriador colado na parte superior do bloco do chip. Para simplificar, iremos estudar o caso unidimensional, analisando apenas a seção transversal do chip, considerando que a espessura do chip é muito fina e que a variação de temperatura ocorre apenas na direção x.

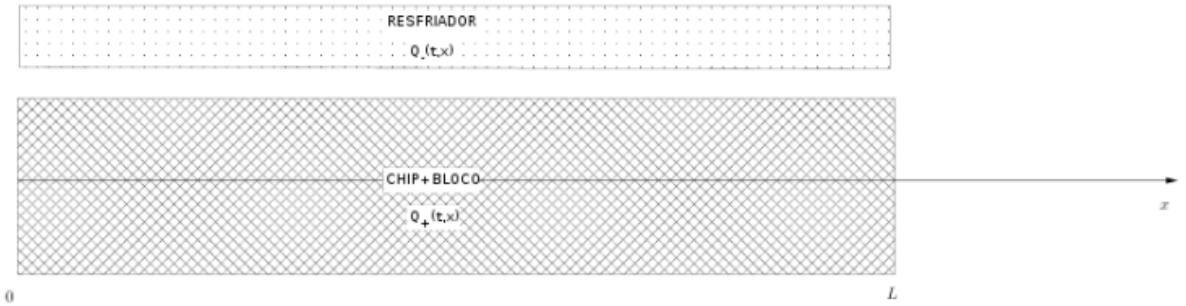


Figura 1: retirada do enunciado da tarefa 3

A equação a ser estudada é a equação de calor, fornecida no enunciado e mostrada abaixo:

$$\rho C \frac{\partial T(t,x)}{\partial t} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial T(t,x)}{\partial x}\right) + Q(t,x), \tag{1}$$

onde

- $T(t,x)$ é a temperatura do chip na posição x e instante de tempo t,
- ρ é a densidade do material do chip (exemplo: o silício tem densidade $\rho = 2300kg/m^3$),
- C é o calor específico do material (exemplo: o calor específico do silício é $C = 750J/Kg/K$),
- k é o parâmetro de condutividade térmica do material (exemplo: o silício tem condutividade de $k = 3,6W/(mK)$).
- Q é uma fonte de calor. É a soma do calor gerado pelo chip (Q_+) com o calor retirado do sistema pelo resfriador (Q_-), tal que $Q = Q_+ - Q_-$.

No caso de um estado estacionário, essa equação se resume a:

$$-\frac{\partial}{\partial x}\left(k(x)\frac{\partial T(x)}{\partial x}\right) = Q(x) \tag{2}$$

Que pode ser resolvida numericamente com uso do método de elementos finitos.

2. Objetivo

O principal objetivo desta tarefa é a solução da equação (2) acima com auxílio do método de elementos finitos.

É conhecido o método de elementos finitos para solucionar equações do tipo:

$$L(u(x)) := (-k(x)u'(x))' + q(x)u(x) = f(x)\ x \in (0,1),\ u(0) = u(1) = 0$$

$$\text{onde } k(x) > 0, q(x) \geq 0, k(x) \in C^1[0,1], q(x), f(x) \in C[0,1].$$

Uma solução clássica dessa equação é u(x) descrita abaixo:

$$u(x) \in V_0 = \{v \in C^2[0,1] : v(0) = v(1) = 0\}$$

Com isso, podemos escrever que:

$$\int_0^1 L(u(x))v(x)\ dx = \int_0^1 f(x)v(x)\ dx$$

O que resulta em:

$$\int_0^1 \left[k(x)u'(x)v'(x) + q(x)u(x)v(x)\right]\ dx = \int_0^1 f(x)v(x)\ dx \quad , \forall v \in U_0$$

E escreve-se:

$$\langle u, v\rangle_L = \int_0^1 \left[k(x)u'(x)v'(x) + q(x)u(x)v(x)\right]\ dx$$

Para solucionar o problema acima, podemos utilizar o método de Rayleigh-Ritz. Este método aproxima a solução u(x) ao minimizar a integral sobre um conjunto menor de funções que são combinações lineares de certas funções de base $\phi_1, \phi_2, ..., \phi_n$ que são linearmente independentes e satisfazem:

$\phi_i(0) = \phi_i(1) = 0, \quad para\ cada\ i = 1, 2, \dots, n.$

Com isso obtém-se uma aproximação de u(x) dada por $\phi(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x)$. Aí ficamos com o seguinte sistema linear:

$$\begin{bmatrix} \langle\phi_1,\phi_1\rangle_L & \langle\phi_2,\phi_1\rangle_L & \dots & \langle\phi_n,\phi_1\rangle_L \\ \dots & \dots & \dots & \dots \\ \langle\phi_1,\phi_n\rangle_L & \langle\phi_2,\phi_n\rangle_L & \dots & \langle\phi_n,\phi_n\rangle_L \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ .. \\ \alpha_n \end{bmatrix} = \begin{bmatrix} \langle f,\phi_1\rangle \\ .. \\ \langle f,\phi_n\rangle \end{bmatrix}$$

Define-se $h = \frac{1}{n+1}$ $e\ x_i = ih,\ i = 0, 1, ..., n+1$ e com isso a base escolhida é dada pelas funções "chapéu" $\phi_i(x)$ que valem 0 fora de $[x_{i-1}, x_{i+1}]$, valem $\phi_i = \frac{x - x_i}{h}$ em $[x_{i-1}, x_i]$ e $\phi_i = \frac{x_{i+1} - x}{h}$ em $[x_i, x_{i+1}]$.

Disso, decorre-se que $\langle\phi_i, \phi_j\rangle_L = 0$ se $|i - j| > 1$. Assim, a matriz é tridiagonal com valores 2/h na diagonal principal e -1/h nas diagonais secundárias.

Para calcular os produtos internos dessa matriz, que são dados por integrais simples, vamos aproximar os valores das integrais utilizando o que foi desenvolvido na atividade 2, que tratava da fórmula de Gauss de 2 pontos. Após montado o sistema tridiagonal, ele será resolvido pela algoritmo LU também desenvolvido previamente na atividade 1 da disciplina.

Para casos de fronteira não homogênea, a seguinte mudança deve ser realizada:

$$L(v(x)) = f(x) + (b-a)k'(x) - q(x)(a + (b-a)x) = \tilde{f}(x) \quad , v(0) = v(1) = 0 \ .$$

Neste caso,
$u(x) = v(x) + a + (b-a)x$ *com condições de fronteira* $u(0) = a$ *e* $u(1) = b$.

Segue a demonstração:

$L(v(x)) = L(u(x)) + L(+a) + L((b-a)x)$
Considerando
$L(u(x)) = f(x)$, $L(+a) = 0$, $L((b-a)x) = k'(x)(b-a) + q(x)(b-a)x$
$e \ \ u(x) = v(x) + a + (b-a)x.$
Daí:
$L(u(x)) = L(v(x)) + L(a) + L((b-a)x)$
O que prova que L(u(x)) é uma transformação linear e que a equação dada para v(x) é de fato uma solução para a equação de métodos finitos utilizada.

Quando o intervalo para a equação diferencial for [0, L], usamos splines lineares neste intervalo com nós igualmente espaçados tomando-se h = L/(n+1) e xi = ih, i = 0, 1, … , n+1.

Podemos obter as expressões da base composta pelas funções chapéu modificadas para condições de contorno não homogêneas ao adaptar o valor de h. Segue a demonstração:

Para $x \in (0,1)$; $u(0) = u(1) = 0$

$$L(u(x)) = (-k(x) \cdot u'(x))' + q(x) \cdot u(x) = f(x)$$

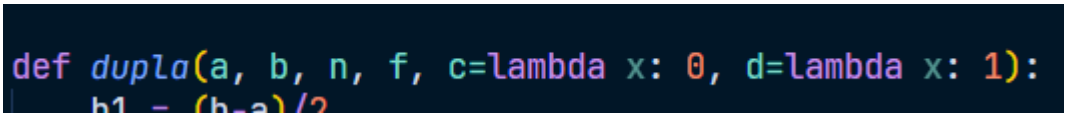
e para $x \in (0, L)$:

$$g(x) = L(u(\frac{x}{L})) = (-k(\frac{x}{L}) \cdot u'(\frac{x}{L}))'$$
$$g(x) = -k'(\frac{x}{L}) \cdot \frac{1}{L} \cdot u'(\frac{x}{L}) - k(\frac{x}{L}) \cdot u"(\frac{x}{L}) \cdot \frac{1}{L}$$
$$g(x) = L(u(\frac{x}{L})) = \frac{L(u(x)}{L}$$

3. Desenvolvimento

O programa desenvolvido nesta atividade deve resolver equações como a equação (4) do enunciado. Utilizou-se no código desta atividade algumas das funções criadas nas atividades anteriores, como as funções que resolvem um sistema linear tridiagonal e as funções que calculam numericamente integrais pelo método de Gauss

Para diminuir a quantidade de parãmetros e simplicar o entendimento do código, alguns valores já foram postos nas funções como "padrão", sendo atribuidos na definição delas. Um exemplo disso é o mostrado na foto abaixo, onde o limite de integração padrão foi definido como sendo de 0 a 1:



Abaixo seguem as funções utilizadas nesta atividade:

```
import numpy as np
from math import e

n6   =   [(-0.23861918608319690863050 17,   0.46791393457269104738987 03),
(-0.66120938646626451366139 96,             0.36076157304813860756983 35),
(-0.93246951420315202781230 16, 0.17132449237917034504029 61),
           (0.23861918608319690863050 17,   0.46791393457269104738987 03),
(0.66120938646626451366139 96,              0.36076157304813860756983 35),
(0.93246951420315202781230 16, 0.17132449237917034504029 61)]

n8   =   [(-0.18343464249564980493947 61,   0.36268378337836198296515 04),
(-0.52553240991632898581773 90,             0.31370664587788728733796 22),
(-0.79666647741362673959155 39,             0.22238103445337447054435 60),
(-0.96028985649753623168356 09, 0.10122853629037625915253 14),
           (0.18343464249564980493947 61,   0.36268378337836198296515 04),
(0.52553240991632898581773 90,              0.31370664587788728733796 22),
(0.79666647741362673959155 39,              0.22238103445337447054435 60),
(0.96028985649753623168356 09, 0.10122853629037625915253 14)]

n10   =   [(-0.14887433898163121088482 60,   0.29552422471475287017389 30),
(-0.43339539412924719079926 59,             0.26926671930999635509122 69),
(-0.67940956829902440623432 74,             0.21908636251598204399553 49),
(-0.86506336668898451073209 67,             0.14945134915058059314577 63),
(-0.97390652851717172007796 40, 0.06667134430868813759356 88),
           (0.14887433898163121088482 60,   0.29552422471475287017389 30),
(0.43339539412924719079926 59,              0.26926671930999635509122 69),
(0.67940956829902440623432 74,              0.21908636251598204399553 49),
(0.86506336668898451073209 67,              0.14945134915058059314577 63),
(0.97390652851717172007796 40, 0.06667134430868813759356 88)]
```

```
dimensoes = [7, 15, 31, 63]


def tridiagonalLU(n, a, b, c, cyclic):

    if (cyclic):
        a = a[:n - 1]
        a[0] = 0
        b = b[:n - 1]
        c = c[:n - 1]
        c[n - 2] = 0
        n = n - 1

    L = np.zeros((n, n))
    U = np.zeros((n, n))
    u = np.zeros((n))
    l = np.zeros((n))
    u[0] = b[0]

    for i in range(1, n):
        l[i] = a[i]/u[i-1]
        u[i] = b[i] - l[i]*c[i-1]

    for i in range(0, n):
        L[i][i] = 1
        U[i][i] = u[i]
        if i < n - 1:
            L[i+1][i] = l[i+1]
            U[i][i+1] = c[i]

    return L, U


def initializeResults(n, manual=True):
    array = np.empty((n))
    if(manual):
        for i in range(0, n):
            array[i] = input(f"insira d{str(i + 1)}: ")

        return array

    for position in range(1, n + 1):
```

```
        array[position - 1] = np.cos((2 * np.pi * position**2)/(n**2))

    print("O vetor contendo os valores d obtido é: ", array, "\n")

    return array


def tridiagonalSolution(n, a, b, c, d):  # Ax = d
    y = np.zeros((n))
    x = np.zeros((n))
    # Ly = d
    L, U = tridiagonalLU(n, a, b, c, False)

    y[0] = d[0]
    for i in range(1, n):
        y[i] = d[i]-L[i][i - 1]*y[i-1]

    # Ux = y
    x[n-1] = y[n-1]/U[n-1][n-1]
    for i in range(n-2, -1, -1):
        x[i] = (y[i]-U[i][i+1]*x[i+1])/U[i][i]

    return x


def dupla(a, b, n, f, c = lambda x: 0, d = lambda x: 1):
    h1 = (b-a)/2
    h2 = (b+a)/2
    J = 0

    for i in range(len(n)):
        JX = 0
        r1 = n10[i][0]
        w1 = n10[i][1]
        x = h1*r1 + h2
        c1 = c(x)  # definir c(x) para cada exemplo
        d1 = d(x)  # definir d(x) para cada exemplo
        k1 = (d1 - c1)/2
        k2 = (d1 + c1)/2

        for j in range(len(n)):
            r2 = n10[j][0]
            w2 = n10[j][1]
```

```
            y = k1*r2 + k2
            Q = f(x, y)
            JX = JX + w2*Q

        J = J + w1*k1*JX

    J = h1*J
    return(J)


def innerProduct(anterior, atual, proximo, h, n, case, f, k, q):
    if case == "inferior":
        return (-1) * ((1 / h) ** 2) * dupla(anterior, atual, n, lambda
x, y: k(x, y) + (atual - x) * (x - anterior) * q(x, y))

    if case == "superior":
        return (-1) * ((1 / h) ** 2) * dupla(atual, proximo, n, lambda
x, y: (k(x, y) + (proximo - x) * (x - atual) * q(x, y)))

    if case == "principal":
        return (1 / h) ** 2 * (dupla(anterior, atual, n, lambda x, y:
(k(x, y) + (x - anterior)**2 * q(x, y))) + dupla(atual, proximo, n,
lambda x, y: (k(x, y) + (proximo - x)**2 * q(x, y))))

    if case == "resultados":
        return (1 / h) * (dupla(anterior, atual, n, lambda x, y: ((x -
anterior) * f(x, y))) + dupla(atual, proximo, n, lambda x, y: ((proximo
- x) * f(x, y))))

    return 0


def solveSystem(f, k, q, L, dim, n):
    h = L / (dim + 1)
    xi = [i * h for i in range(dim + 2)]
    a = list()
    b = list()
    c = list()
    d = list()

    for i in range(dim):
        anterior = xi[i]
        atual = xi[i + 1]
```

```
        proximo = xi[i + 2]
        a.append(innerProduct(anterior, atual,
                 proximo, h, n, "inferior", f, k, q))
        b.append(innerProduct(anterior, atual,
                 proximo, h, n, "principal", f, k, q))
        c.append(innerProduct(anterior, atual,
                 proximo, h, n, "superior", f, k, q))
        d.append(innerProduct(anterior, atual,
                 proximo, h, n, "resultados", f, k, q))

    return tridiagonalSolution(dim, a, b, c, d)


def phi(x, xi_anterior, xi_atual, xi_proximo, h):
    if (xi_anterior < x <= xi_atual):
        return (x - xi_anterior)/h

    if (xi_atual < x <= xi_proximo):
        return (xi_proximo - x)/h

    return 0


def uBarra(x, xi, solucoes, h, homogeneo=True, a=1, b=1, L=1):
    if(homogeneo):
        uBarra = 0
        for i in range(len(solucoes)):
            uBarra += solucoes[i] * phi(x, xi[i], xi[i + 1], xi[i + 2],
h)
        return uBarra
    return uBarra(x, xi, solucoes, h) + (a + (b - a) * x) / L


def biggestError(solucoes, u, dim, L):
    h = L / (dim + 1)
    xi = [i * h for i in range(dim + 2)]
    erroMax = 0
    erroMaxUExato = 0
    erroMaxUBarra = 0
    for x in xi:
        if abs(u(x) - uBarra(x, xi, solucoes, h)) > erroMax:
            erroMax = abs(u(x) - uBarra(x, xi, solucoes, h))
            erroMaxUExato = u(x)
```

```
            erroMaxUBarra = uBarra(x, xi, solucoes, h)
    return erroMax, erroMaxUExato, erroMaxUBarra
```

3.1 Validação:

No primeiro passo da atividade, nosso programa resolve a equação (4) do enunciado utilizando método de elementos finitos com q(x) = 0, no intervalo [0, L] e condições de contorno u(0) = a, u(L) = b. Escolheu-se também o valor n10 no método de Gauss, para obter melhor aproximação do valor real das integrais calculadas pelo método de integração numérica.

O primeiro teste é feito com k(x) = 1, q(x) = 0, f(x) = 12x(1-x) - 2, condições de contorno homogêneas. A solução exata é $u(x) = x^2(1-x)^2$.

Como complemento para a validação, testa-se a solução do programa para o intervalo [0, 1] onde $k(x) = e^x$, $q(x) = 0$, $f(x) = e^x + 1$, com condições de contorno homogêneas. A solução exata é $u(x) = (x-1)(e^{-x} - 1)$.

Para o modo 1, o procedimento é como mostrado abaixo:

```
def main():
    print("")
       print("Bem-vindo ao programa de Modelagem de um Sistema de
Resfriamento de Chips escrito por Arthur Maia e Caio Balreira. \n")
    print("Neste programa, existem 3 funcionalidades:\n 1) Validação do
Método dos Elementos Finitos \n 2) Equilíbrio com Forçantes de Calor \n
3) Equilíbrio com variação de material \n 0) Encerrar o programa \n")

    while True:
        while True:
            try:
                choice = int(
                            input("Escolha a funcionalidade que deseja
executar: "))
                print("")
                if (choice < 0 or choice > 3):
                    raise ValueError()
                break
            except ValueError:
                    print("Esta funcionalidade não é válida. Por favor,
tente outra \n")

        if (choice == 1):

            print(
```

```
                "Para a validação com k(x) = 1, q(x) = 0 e f(x) = 12x(1
- x) - 2, com intervalo [0, 1]: \n")
            print("Analisando os erros máximos para diferentes valores
de n:")

            for n in dimensoes:
                print(f"\n\nPara n = {n}:\n")
                solucoes = solveSystem(
                     lambda x, y: 12 * x * (1 - x) - 2, lambda x, y: 1,
lambda x, y: 0, 1, n, n10)
                erroMax, erroMaxUExato, erroMaxUBarra = biggestError(
                    solucoes, lambda x: x**2 * (1 - x)**2, n, 1)
                print(
                      f" Valor do u barra: {erroMaxUBarra}\n Valor do u
exato: {erroMaxUExato} \n Erro encontrado: {erroMax}")

            print(
                 "\n\nPara a validação com k(x) = e^x, q(x) = 0 e f(x) =
e^x + 1, com intervalo [0, 1]: \n")
            print("Analisando os erros máximos para diferentes valores
de n:")

            for n in dimensoes:
                print(f"\n\nPara n = {n}:\n")
                solucoes = solveSystem(
                     lambda x, y: e**x + 1, lambda x, y: e**x, lambda x,
y: 0, 1, n, n10)
                erroMax, erroMaxUExato, erroMaxUBarra = biggestError(
                    solucoes, lambda x: (x - 1)*(e**(-x) - 1), n, 1)
                print(
                      f" Valor do u barra: {erroMaxUBarra}\n Valor do u
exato: {erroMaxUExato} \n Erro encontrado: {erroMax}")
```

E os resultados obtidos foram:

```
Para a validação com k(x) = 1, q(x) = 0 e f(x) = 12x(1 - x) - 2, com intervalo [0, 1]:

Analisando os erros máximos para diferentes valores de n:


Para n = 7:

 Valor do u barra: 0.011962890624999998
 Valor do u exato: 0.011962890625
 Erro encontrado: 1.734723475976807e-18


Para n = 15:

 Valor do u barra: 0.0605621337890624
 Valor do u exato: 0.0605621337890625
 Erro encontrado: 9.71445146547012e-17


Para n = 31:

 Valor do u barra: 0.062012672424316 05
 Valor do u exato: 0.062012672424316406
 Erro encontrado: 3.5388358909926865e-16


Para n = 63:

 Valor do u barra: 0.04615783691406208
 Valor do u exato: 0.0461578369140625
 Erro encontrado: 4.2327252813834093e-16
```

```
Para a validação com k(x) = e^x, q(x) = 0 e f(x) = e^x + 1, com intervalo [0, 1]:

Analisando os erros máximos para diferentes valores de n:


Para n = 7:

 Valor do u barra: 0.19534565389865072
 Valor do u exato: 0.19544420075564234
 Erro encontrado: 9.854685699162147e-05


Para n = 15:

 Valor do u barra: 0.199297888768106 64
 Valor do u exato: 0.19932270388431073
 Erro encontrado: 2.4815116204085497e-05


Para n = 31:

 Valor do u barra: 0.1982210201687935
 Valor do u exato: 0.19822723114480412
 Erro encontrado: 6.210976010612157e-06


Para n = 63:

 Valor do u barra: 0.1989798148463074
 Valor do u exato: 0.19898136848366316
 Erro encontrado: 1.5536373557667 46e-06
```

Nota-se que os erros para os valores de u barra calculados aumentam quanto maior for o valor de n.

3.2 Equilíbrio com forçantes de calor

Nesta seção, solicita-se o seguinte:

Vamos considerar que o chip seja formado apenas de silício ($k(x) = k = 3,6W/(mK)$), considerando que há produção de calor pelo chip e que exista resfriamento. Vamos assumir que o chip esquenta mais em sua parte central que nas bordas, o que pode ser modelado por uma Gaussiana da seguinte forma,

$$Q_+(x) = Q_+^0 e^{-(x-L/2)^2/\sigma^2} \tag{9}$$

com Q_+^0 uma constante indicando o máximo de calor gerado no centro do chip e σ controlando a variação de geração de calor em torno do ponto central do chip. Se σ for muito pequeno, podemos ter uma calor gerado praticamente somente no centro do chip.

Quanto ao resfriamento, podemos modelar de forma análoga, ou, por exemplo, assumir que o resfriamento se dá de forma uniforme ($Q_-(x) = Q_-^0$ constante), ou ainda que o resfriamento seja mais intenso próximo dos extremos, usando

$$Q_-(x) = Q_-^0 \left(e^{-(x)^2/\theta^2} + e^{-(x-L)^2/\theta^2} \right). \tag{10}$$

Use o seu código de elementos finitos para simular algumas situações de equilíbrio variando parâmetros do modelo conforme considere adequado à aplicação real. Comece pelo caso mais simples, com calor gerado e retirado constantes, e vá acrescentando complexidade. Relate o que observou no relatório.

Retirado diretamente do enunciado. As funções escolhidas para serem modeladas foram:

- Q(x) = 1, por ser simples
- Q(x) = -x^2, por ser negativa, assim representando um resfriamento
- Q(x) = sin(10x), por ser periódica

O código para este modo é como mostrado abaixo:

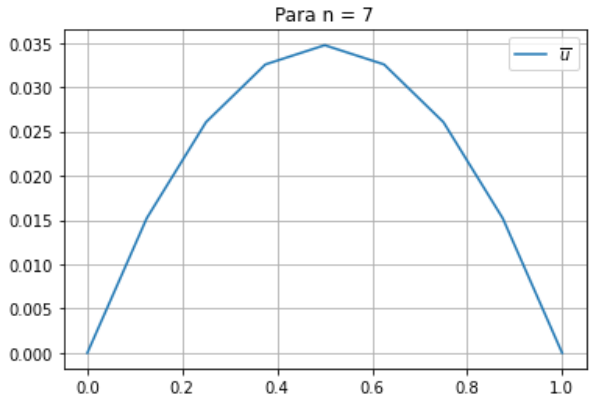
```
 if (choice == 2):
            print(
                  "Para o Equilíbrio com Forçantes de Calor no intervalo
[0, 1]: \n")
            print("Primeira situação: Q(x) = 1: \n")

            for n in dimensoes:
                print(f"Para n = {n}:\n")

                solucoes = solveSystem(
                       lambda x, y: 1, lambda x, y: 3.6, lambda x, y: 0,
1, n, n10)

                h = 1/(n + 1)
                xi = [i * h for i in range(n + 2)]
                barras = list()
                for x in xi:
                    barras.append(uBarra(x, xi, solucoes, h))
                    print(
```

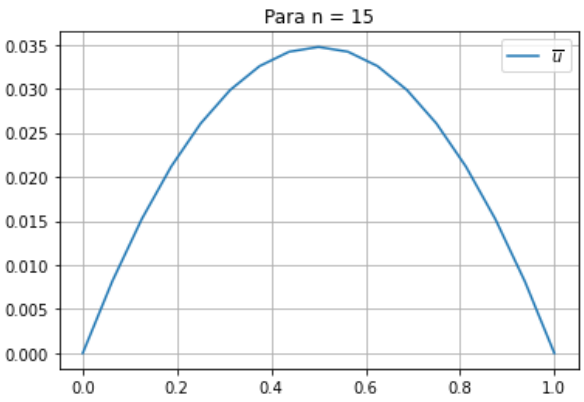
```
                        f"x = {x}; u barra = {uBarra(x, xi, solucoes,
h)}")

                print("\n\n")
                plt.plot(xi, barras, label="$\overline{u}$")
                plt.title(f"Q(x) = 1 para n = {n}")
                plt.legend()
                plt.grid()
                plt.show()

            print("Segunda situação: Q(x) = -x^2: \n")

            for n in dimensoes:
                print(f"Para n = {n}:\n")

                solucoes = solveSystem(
                      lambda x, y: (-1)*(x**2), lambda x, y: 3.6, lambda
x, y: 0, 1, n, n10)

                h = 1/(n + 1)
                xi = [i * h for i in range(n + 2)]
                barras = list()
                for x in xi:
                    barras.append(uBarra(x, xi, solucoes, h))
                    print(
                           f"x = {x}; u barra = {uBarra(x, xi, solucoes,
h)}")

                print("\n\n")
                plt.plot(xi, barras, label="$\overline{u}$")
                plt.title(f"Q(x) = -x^2 para n = {n}")
                plt.legend()
                plt.grid()
                plt.show()

            print("Terceira situação: Q(x) = sin(10x): \n")

            for n in dimensoes:
                print(f"Para n = {n}:\n")

                solucoes = solveSystem(
                     lambda x, y: np.sin(10*x), lambda x, y: 3.6, lambda
x, y: 0, 1, n, n10)
```

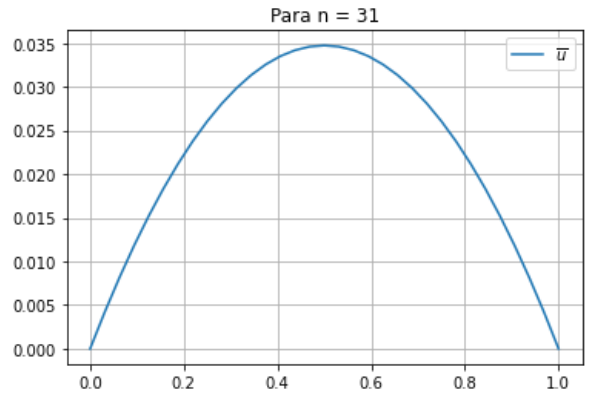
```
                h = 1/(n + 1)
                xi = [i * h for i in range(n + 2)]
                barras = list()
                for x in xi:
                    barras.append(uBarra(x, xi, solucoes, h))
                    print(
                           f"x = {x}; u barra = {uBarra(x, xi, solucoes,
h)}")


                print("\n\n")
                plt.plot(xi, barras, label="$\overline{u}$")
                plt.title(f"Q(x) = sin(10x) para n = {n}")
                plt.legend()
                plt.grid()
                plt.show()
```

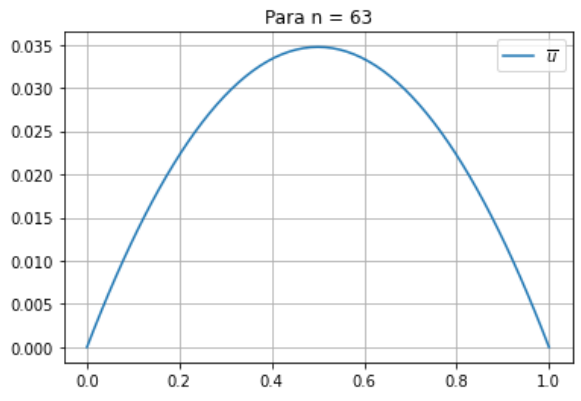
Os resultados obtidos para esta etapa são mostrados abaixo:

```
Para o Equilíbrio com Forçantes de Calor no intervalo [0, 1]:

Primeira situação: Q(x) = 1:
```

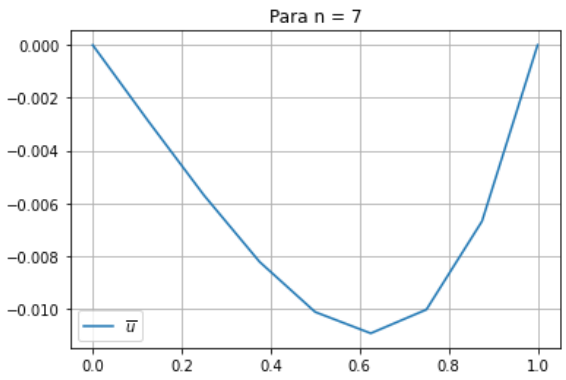


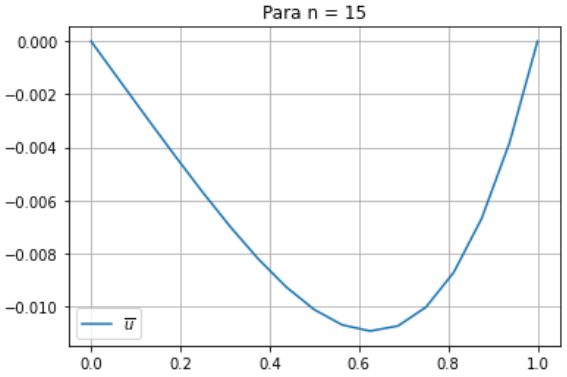


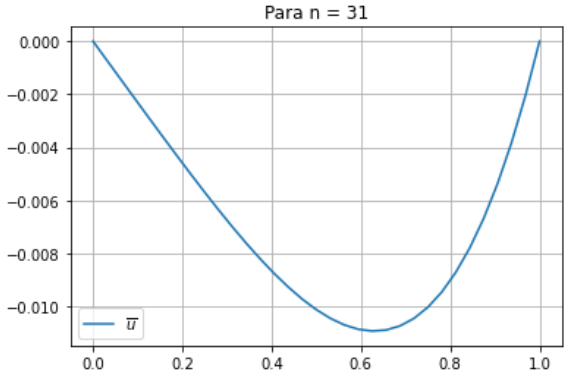


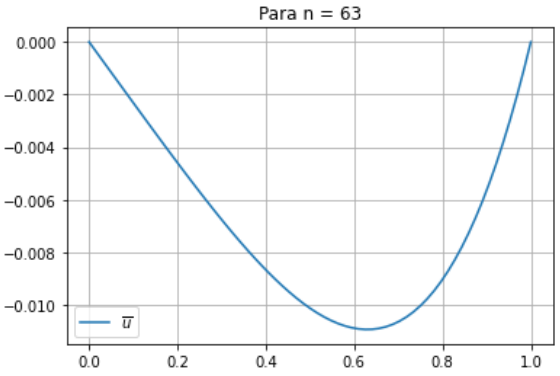


Segunda situação: Q(x) = -x^2:

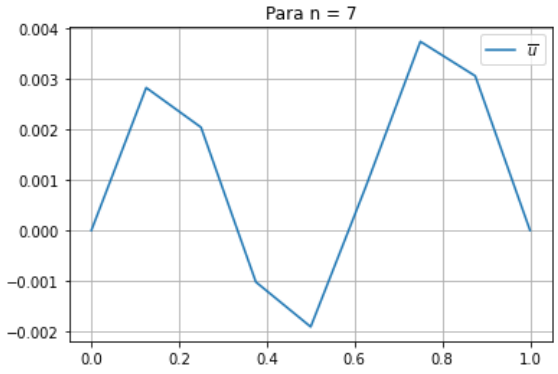


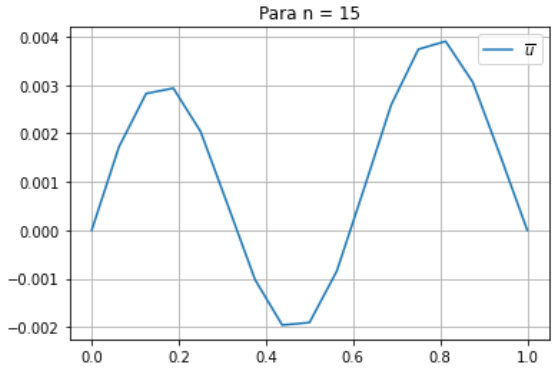


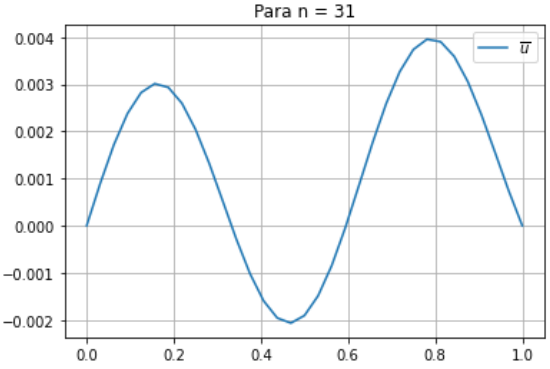


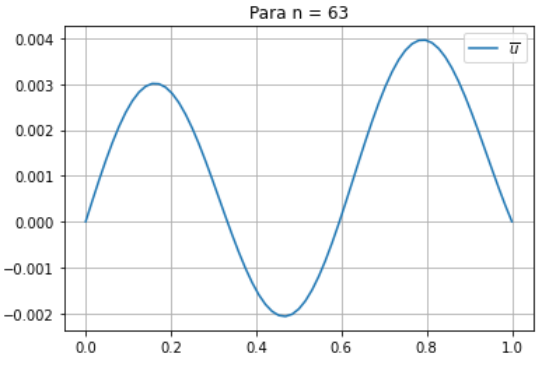


Terceira situação:Q(x) = sin(10x):









Percebe-se que o erro decai como esperado. Além disso, nota-se que para n maiores, a curva fica mais “suave”, indicando melhor aproximação do resultado exato.

3.3 Equilíbrio com variação de material

Nesta seção, solicita-se o seguinte:

Suponha agora que no bloco do processador tenhamos o chip, formado de silício, envolto por outro material. Isso faz com que k dependa de x, por exemplo como

$$k(x) = \begin{cases} k_s, \text{se } x \in (L/2 - d, L/2 + d), \\ k_a, \text{caso contrário,} \end{cases} \tag{11}$$

sendo k_s a condutividade térmica do silício e k_a a do material que envolve o chip e forma o bloco.

Usando o seu código de elementos finitos você pode verificar, por exemplo, o que acontece se o material que envolve o chip for alumínio ($k_a = 60W/mK$), ou outros materiais. Inclua essa análise no relatório.

Retirado diretamente do enunciado. As funções escolhidas para serem modeladas foram:

- Q(x) = 1
- Q(x) = -x^2
- Q(x) = sin(10x)

Pelos mesmos motivos citados anteriormente. O código para este modo é como mostrado abaixo:

```
if (choice == 3):
    print(
        "Para o Equilíbrio com Variação de Material no intervalo [0, L]: \n")

    L = int(input("Insira o tamanho do chip (L): "))
    d = int(input("Insira o raio da parte de silício (d): "))
    ka = int(
        input("Insira o parâmetro de condutividade térmica do material (k): "))

    def k(x, y, L, d):
        if (L/2 - d) < x < (L/2 + d):
            return 3.6
        return ka
```

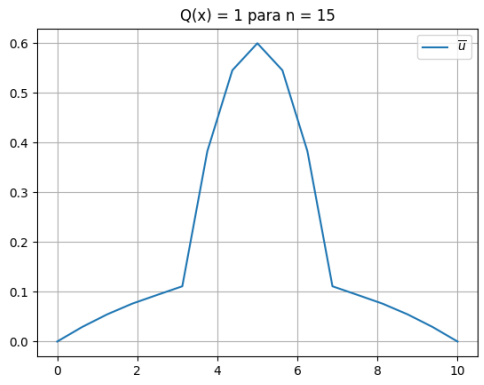
```
print("Primeira situação: Q(x) = 1: \n")

for n in dimensoes:
    print(f"Para n = {n}:\n")

    solucoes = solveSystem(
        lambda x, y: 1, lambda x, y: k(x, y, L, d), lambda x, y: 0, L, n, n10)

    h = L/(n + 1)
    xi = [i * h for i in range(n + 2)]
    barras = list()
    for x in xi:
        barras.append(uBarra(x, xi, solucoes, h))
        print(
            f"x = {x}; u barra = {uBarra(x, xi, solucoes, h)}")

    print("\n\n")
    plt.plot(xi, barras, label="$\overline{u}$")
    plt.title(f"Q(x) = 1 para n = {n}")
    plt.legend()
    plt.grid()
    plt.show()
```

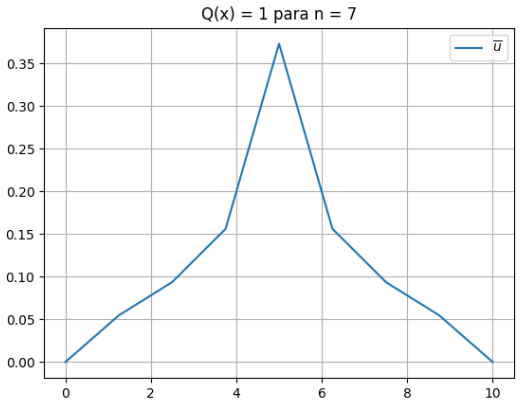
```
print("Segunda situação: Q(x) = -x^2: \n")

for n in dimensoes:
    print(f"Para n = {n}:\n")

    solucoes = solveSystem(
        lambda x, y: (-1)*(x**2), lambda x, y: k(x, y, 1, 0.25), lambda x, y: 0, 1, n, n10)

    h = 1/(n + 1)
    xi = [i * h for i in range(n + 2)]
    barras = list()
    for x in xi:
        barras.append(uBarra(x, xi, solucoes, h))
        print(
            f"x = {x}; u barra = {uBarra(x, xi, solucoes, h)}")

    print("\n\n")
    plt.plot(xi, barras, label="$\overline{u}$")
    plt.title(f"Q(x) = -x^2 para n = {n}")
    plt.legend()
    plt.grid()
    plt.show()
```

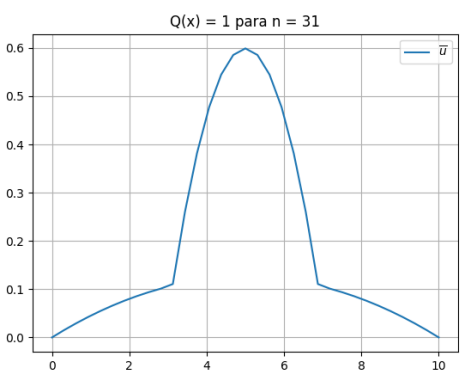
```python
print("Terceira situação: Q(x) = sin(10x): \n")

for n in dimensoes:
    print(f"Para n = {n}:\n")

    solucoes = solveSystem(
        lambda x, y: np.sin(10*x), lambda x, y: k(x, y, 1, 0.25), lambda x, y: 0, 1, n, n10)

    h = 1/(n + 1)
    xi = [i * h for i in range(n + 2)]
    barras = list()
    for x in xi:
        barras.append(uBarra(x, xi, solucoes, h))
        print(
            f"x = {x}; u barra = {uBarra(x, xi, solucoes, h)}")

    print("\n\n")
    plt.plot(xi, barras, label="$\overline{u}$")
    plt.title(f"Q(x) = sin(10x) para n = {n}")
    plt.legend()
    plt.grid()
    plt.show()
```

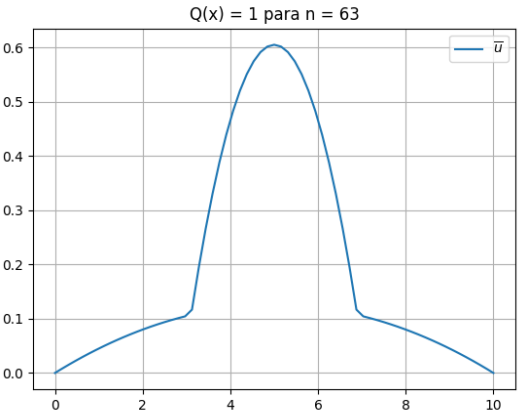
Os resultados obtidos para esta etapa são mostrados abaixo:

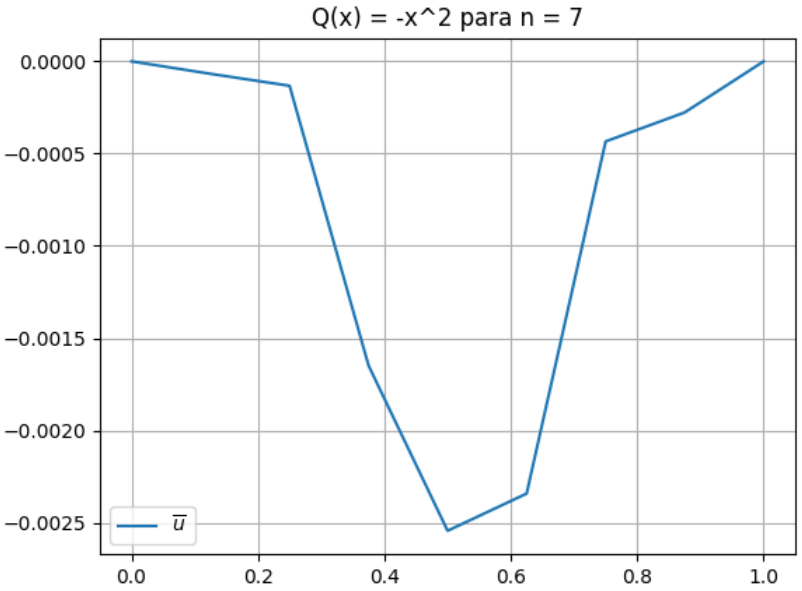
```
Para o Equilíbrio com Variação de Material no intervalo [0, L]:

Insira o tamanho do chip (L): 10
Insira o raio da parte de silício (d): 2
Insira o parâmetro de condutividade térmica do material (k): 100
Primeira situação: Q(x) = 1:
```

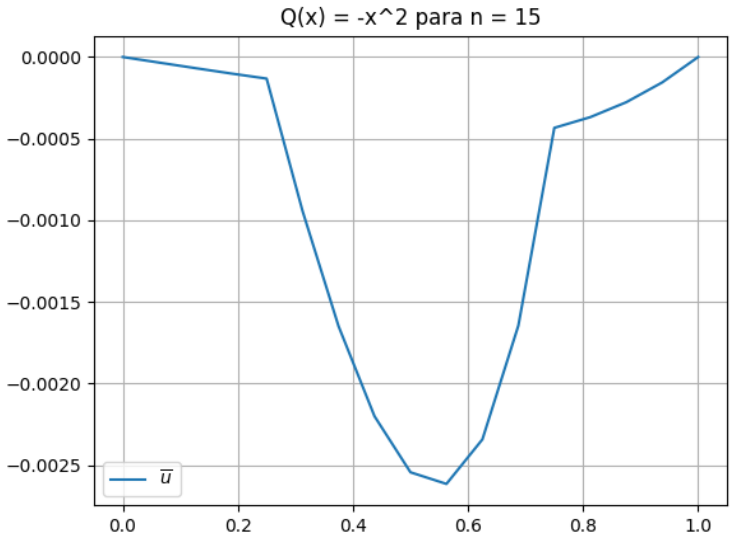


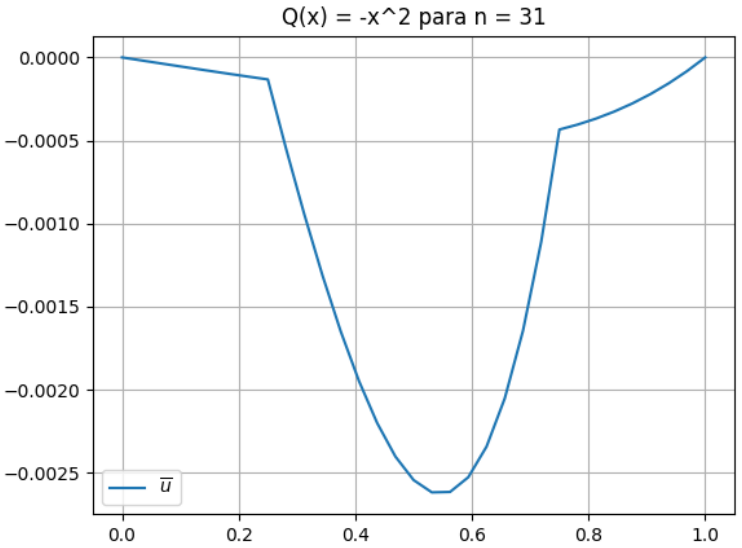


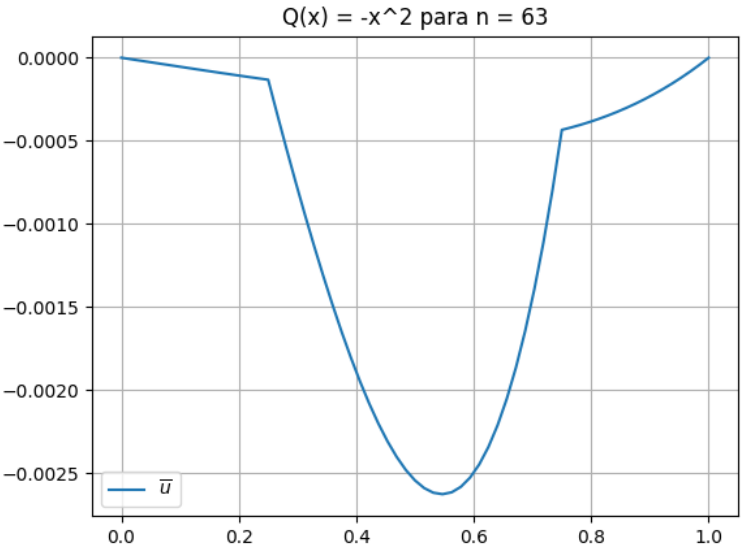


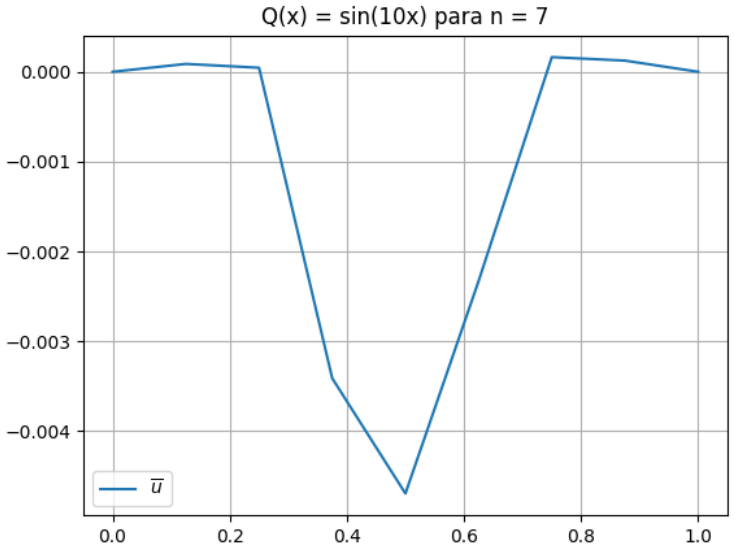


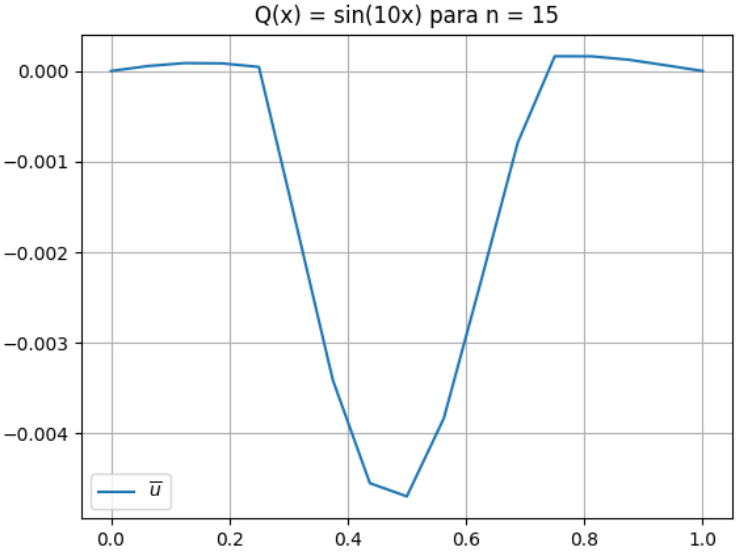

Q(x) = -x^2 para n = 7
0.0000
−0.0005
−0.0010
−0.0015
−0.0020
−0.0025
$\overline{u}$
0.0
0.2
0.4
0.6
0.8
1.0

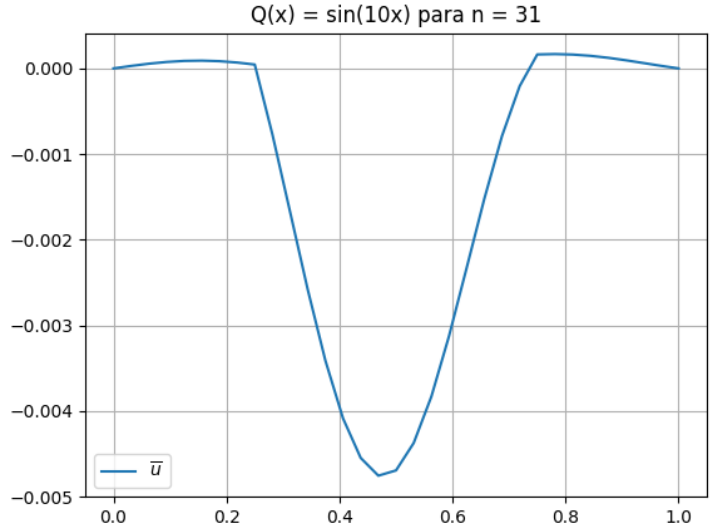

Q(x) = -x^2 para n = 15
0.0000
−0.0005
−0.0010
−0.0015
−0.0020
−0.0025
$\overline{u}$
0.0
0.2
0.4
0.6
0.8
1.0

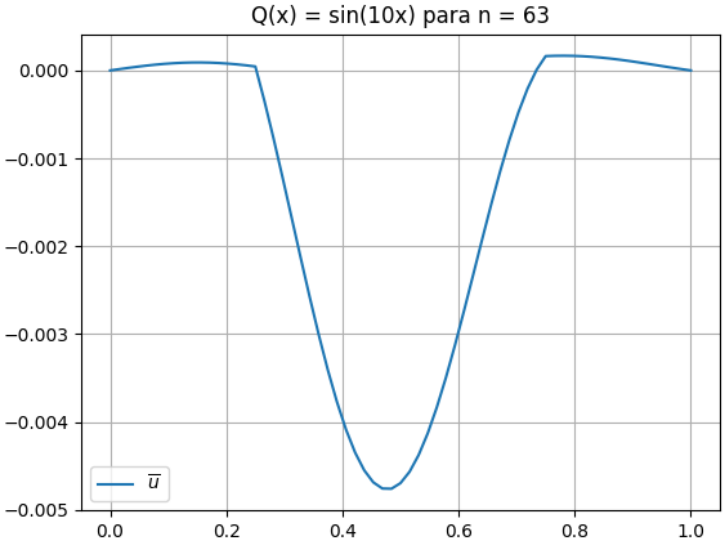

Q(x) = -x^2 para n = 31
0.0000
−0.0005
−0.0010
−0.0015
−0.0020
−0.0025
$\overline{u}$
0.0
0.2
0.4
0.6
0.8
1.0


Q(x) = -x^2 para n = 63
0.0000
−0.0005
−0.0010
−0.0015
−0.0020
−0.0025
$\overline{u}$
0.0
0.2
0.4
0.6
0.8
1.0


Q(x) = sin(10x) para n = 7
0.000
−0.001
−0.002
−0.003
−0.004
$\overline{u}$
0.0
0.2
0.4
0.6
0.8
1.0


Q(x) = sin(10x) para n = 15
0.000
−0.001
−0.002
−0.003
−0.004
$\overline{u}$
0.0
0.2
0.4
0.6
0.8
1.0





4. Conclusão

Após a realização deste exercício-programa, pode-se aprender mais sobre o método dos elementos finitos, assim como o método de Rayleigh-Ritz para a resolução de equações diferenciais parciais além de como implementá-los computacionalmente. Também pode-se aprender mais sobre a distribuição de calor em um sólido quando há dissipação e como a distribuição original afeta o calor e como o tamanho do chip e condutividade térmica afetam a distribuição e dispersão de calor no material.